A Scalable Data Analysis Platform for Metagenomics

Wei Tang,* Jared Wilkening,* Narayan Desai,*† Wolfgang Gerlach,†* Andreas Wilke,*† Folker Meyer*†

*Argonne National Laboratory, Argonne, IL, USA
†University of Chicago, Chicago, IL, USA

{wtang, jared, desai, wgerlach, wilke, folker}@mcs.anl.gov

***Abstract*—With the advent of high-throughput DNA sequencing technology, the analysis and management of the increasing amount of biological sequence data has become a bottleneck for scientific progress. For example, MG-RAST, a metagenome annotation system serving a large scientific community worldwide, has experienced a sustained, exponential growth in data submissions for several years; and this trend is expected to continue. To address the computational challenges posed by this workload, we developed a new data analysis platform, including a data management system (Shock) for biological sequence data and a workflow management system (AWE) supporting scalable, fault-tolerant task and resource management. Shock and AWE can be used to build a scalable and reproducible data analysis infrastructure for upper-level biological data analysis services.**

***Index Terms*—data management system, workflow, metagenomics, bioinformatics, data analysis platform, cloud computing**

I. Introduction

Next-generation DNA sequencing technology has dramatically reduced the cost of data generation [23]. This has been greatly benefited genetic biosciences, especially metagenomics. Metagenomics [10] is the study of microbial communities. Insight into these communities sheds light on a wide range of areas, including human health [6] and environmental processes [2]. It typically needs additional depth of sequencing; thus a low sequencing cost is beneficial. In the meantime, however, the computational costs of analysis have grown unsustainably as large amounts of data have been generated. As a result, the bottleneck in metagenomics has moved from sample collection and data generation to data analysis [14].

MG-RAST [4][20] is metagenomics analysis server, providing automated data analysis for metagenome data sets submitted by users via web interfaces. MG-RAST was launched in 2007 at Argonne National Laboratory as a free service and has grown to become a dominant community resource for metagenomic data analysis. As of late June 2013, MG-RAST has analyzed 28 Tbp (tera base pairs, 10^{12} base pairs[1]) of sequence data across 83,000 metagenome samples for over 10,000 registered users from over 60 different countries. The MG-RAST project has experienced the data deluge firsthand, as submission volumes have grown continuously (Figure 1).

This data growth has caused several difficulties. First, computational budgets are effectively flat, and data is growing faster than the increase in computational capacity from Moore's law provides. Second, many bioinformatics tools used for data analysis perform poorly with large-scale data. Third, data management operations, including storage, update, retrieval, and compression, have become more expensive and inconvenient. Last but not least, as the computation is more expensive for larger data size, duplicate computations have become unaffordable. These problems have combined to result in a situation that impacts our ability to run a resource such as MG-RAST.

With these problems, we are motivated to build a new software infrastructure with the goals of providing a scalable, portable, customizable, reusable, and reproducible data analysis capability. To this end, we have developed two software products: Shock and AWE. Specifically, Shock is an object-based data management system for the biological data; it implements metadata management and domain-specific sub-object operations. AWE is a workflow management system supporting flexible, fault-tolerant task management and dynamic scalability for computing clients. AWE and Shock can collectively be used to build a data management and analysis platform for upper level services involving big data, including MG-RAST.

II. Systems and Methodology

A. Motivating Problems and Design Goals

The major problems motivating our design are twofold. First, the data deluge is causing longer job turnaround time as the computing capability is limited within one computing facility. Thus, a remotely scalable architecture would be helpful for increasing the throughput when the system is overloaded.

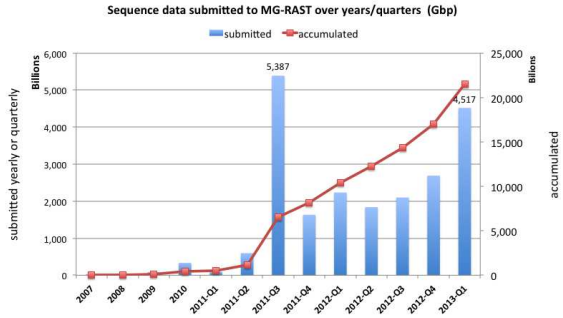

Fig. 1. Data submitted to MG-RAST is continuously growing. (The data burst in Q3/2011 is caused by the data submission from NIH Human Microbiome Project (HMP) [6])

[1]Base pairs are building blocks of the DNA double helix. In the sequence data file, a base pair is represented by a character (one of ACGT or other letters for ambiguity).

Second, with the growing data not managed properly, unnecessary duplicated computation is inevitable, and reproducing of the results become harder. Therefore, a data management ecosystem is needed to help with data sharing and reusing.

To these ends, we designed a new system in which data and computation are loosely coupled in a data management system (Shock) and a workflow management system (AWE).

B. Shock Data Management System

Shock is a data management system for moderate to large-scale biological data sets. Shock is designed as a layer of system software built on top of backend storage systems, providing an object-based storage interface tailored for bioinformatics workloads. An object in Shock encapsulates the actual data file and the metadata of its scientific provenance as well as the computational provenance. Shock supports a number of common bioinformatics data formats and can provide subselection and merge capabilities that minimize I/O requirements for analysis workloads and improve scalability for large numbers of computing clients. These capabilities are implemented by indexing the data in chunks (e.g., one sequence per chunk for biological sequence data).

Shock comprises two major components. One is a web service that provides REST [15] APIs for data storage, retrieval, query, indexing, and so on. The other is backend data storage facilities to handle the storage and retrieval of data files and to operate the data objects such as updating metadata and creating indexes. The object information are stored in MongoDB [5] for the convenience of data query. The actual data file is stored in the backend file systems. Currently Shock supports standard posix file systems; it can be built on top of any parallel or distributed file systems.

To retrieve a data file, one can use a REST API providing the Id of the data. The query functionality enables users to retrieve a set of data that matches a query. For example, one can get all the sequence data from project "X" that has the metadata attribute "origin" equal to "deep sea water." With the metadata support and query functionality, the sharing and reusing existing data become possible and convenient.

Using indexing, one can retrieve any range or part of the sequence file (e.g. get sequence numbers 1000-2000 or get the first or second 100MB of the file). Also, an empty Shock object (object without data file) can be created to accept partial files to merge into the single file in that object. This feature enables split and merge input and output data for embarrassingly parallel tasks to be processed by different computing clients.

Shock is implemented in the Go programming language as an open-source project maintained at Github.com [7]. Go is designed (by Google) for writing large-scale system software and thus is suitable for our case.

C. AWE Workflow Management System

AWE is a workflow management system that manages and executes scientific computing workflows or pipelines. AWE models application-related concepts into three hierarchical elements: job, task, and workunit (Figure 2). A job, characterized by its input data, and workflow description (e.g., data dependencies), is parsed into task, A task represents a certain data analysis operation. A task can be split into multiple workunits running the same command on different parts of the input data. AWE manages and executes these elements in a coordinated and automated fashion.

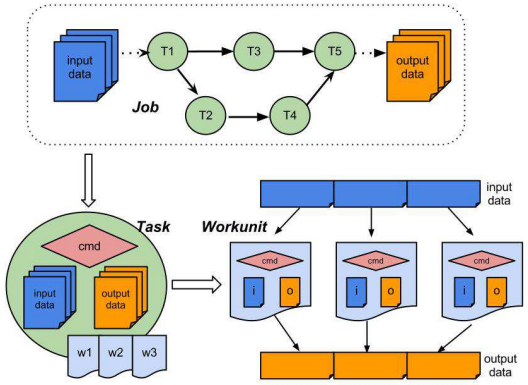

Fig. 2. Key elements managed by AWE: job, task, and workunit.

AWE is designed as a distributed system that contains a centralized server and multiple distributed clients. The server receives job submissions and parses jobs into tasks, splits tasks into workunits, and manages workunits in a queue. The AWE clients, running on distributed, heterogeneous (in terms of hardware or applications) computing resources, keep checking out workunits from the server queue and dispatch the workunits on the local computing resource. AWE uses Shock to handle input and output data (retrieval, storage, splitting, and merge). AWE uses a RESTful [15] API for communication between AWE components and with outside components such as Shock, the job submitter, and the status monitor. These APIs enable incorporation of AWE features into other systems.

A working diagram of AWE is presented in Figure 3. First, the user uploads data to Shock if the data is not already there (0). Then the user can submit job scripts that specify the workflow and the input location (i.e., Shock object ID) to the AWE server (1). Next, the AWE server parses the submitted jobs into tasks and workunits, stores job information in MongoDB, and manages workunits in a queue (2). Meanwhile, the server creates an empty shock object for each workunit to store the output data that will be produced in the future. This empty object may be filled in by multiple cooperative workunits working on different part of the input data (3). The AWE clients keep doing the following repeatedly: obtaining queued workunits (4), retrieving input data from Shock (5), dispatching work on computing resources (6), pushing output data to Shock (7), and notifying the server of the workunit status (8). When the AWE server receives a "work done" notification, it updates the queue by parsing more tasks if their dependent data has become available. When all the tasks of a job are done, the user can download the output data from

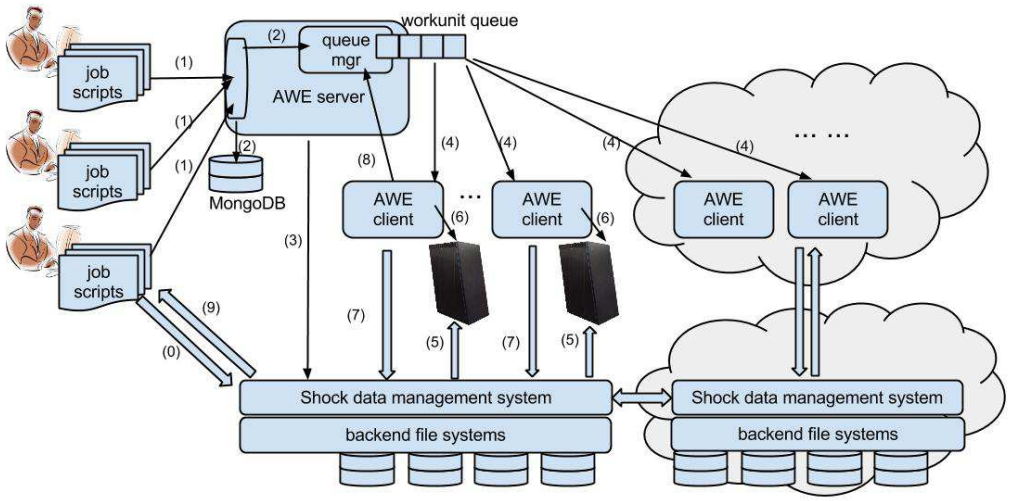

Fig. 3. AWE working diagram. A centralized AWE server parses submitted jobs and manages the workunit queue. Distributed AWE clients repeatedly obtain workunits from the queue and dispatch computation locally. Shock is used for managing the data.

Shock (9).

AWE is also implemented in Go with source code hosted on Github.com [1]. In the following we present some implementation details relevant to our design goals.

Server and clients. The AWE server provides centralized resource management (e.g., queue management and client management) and provides RESTful APIs. AWE clients run as daemons on distributed computing resources. A client comprises workers such as work stealer, data mover, work dispatcher, and heartbeater. These workers run as concurrent threads, and the time of computation and data movement for different workunits can be overlapped.

Splitting tasks. The AWE server is responsible for splitting tasks into workunits by dividing the input data. The data can be divided based on total number of workunits or fixed data size for each workunit. The way to split is configurable. In either way the input data will be divided as evenly as possible. The actual data splitting and merge are done in Shock. AWE only needs to set the index ranges for the input data of each workunit.

Queue management. Queue management is an essential functionality of the AWE server. The workunits in the queue are managed in dynamic, multi-dimensional groups or priorities. Specifically, in order to handle a checkout request, eligible workunits will be selected based on criteria such as group name and supported applications of the requesting client. This approach controls the spatial workunit placement. For the eligible workunits, temporal prioritizing policies will be applied.

Communications. The communication and data transfer between components all use HTTP requests defined in RESTful APIs. The submitted job scripts and checked-out workunits are both defined in JSON format. The communication between the AWE server and client is light weight. During computation, the actual data transfer happens only between Shock and AWE clients.

Fault tolerance and dynamic scalability. AWE provides fault tolerance features to maintain continuous service and support dynamic component come and go. Specifically, the AWE server can automatically recover the queue after restart since the jobs are stored in MongoDB; the AWE clients keep sending heartbeats to the server so that any loss of connection can be detected and corresponding actions will be triggered (e.g., client reregistration when server restarts or workunit requeuing when the hosting client is gone). The AWE client also returns a failed workunit to the server, enabling it to be checked out by other clients. Some unrecoverable failures will result in the suspension of the problematic jobs or clients and notifications to system administrators.

D. Goals Achieved and Use Cases

Scalability and portability. With the data and computation loosely coupled, the tasks can be scaled out to remote resources. One important use case for web-based services such as MG-RAST using AWE is that users contribute their local computing cycles to their own jobs submitted to MG-RAST to achieve fast turnaround.

Task splitting can help parallelize embarrassingly parallel tasks and also make some tools perform better with smaller data sets but it is not a requirement for scaling out tasks.

The scaling-out diagram is suitable for biological data analysis workloads because they are typically of high compute-to-I/O ratio, which means data transfer overhead is negligible compared with the gains in computational expenses.

Reusability and reproducibility. With the provenance information and query capability, Shock and AWE can use previous computed results to save the cost of duplicated computing. For example, when the AWE server detects a task conducts a duplicate computing, it directs the output location to the successive tasks directly. Also, reproducibility

is convenient with the computational provenance information being managed.

Customizability and flexibility. AWE can run any type of pipeline or workflow described in a job script in JSON format to define the data dependencies between tasks and to specify the data location. Task execution and resource management can be very flexible with AWE. The jobs can be allocated based on user authentication so that users with computing clients can checkout their own tasks only. Different tasks within one job can be executed on different machines with different requirements such as hardware configuration, software installation, security group, and data locality. This capability is suitable for bioinformatics pipelines where different stages of the pipeline always have different requirements for software and hardware configurations.

Although our systems are motivated by MG-RAST, they are suitable for building general data management and analysis platforms supporting upper-level services such as Galaxy [16] and Camera [26].

III. Case Study

In this section we present a case study on how Shock and AWE work to run MG-RAST jobs in the cloud environment. We evaluate the scalability and throughput variation using a real MG-RAST workload.

A. Applications and Testbed

For one MG-RAST job that represents a single metagenomic data set, MG-RAST runs a series of data processing/analysis tasks in a pipeline (Figure 4). Tasks are accomplished by bioinformatics tools including our in-house scripts or third-party tools [32].

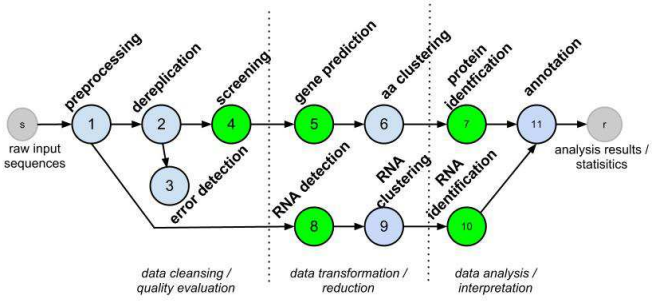

Fig. 4. MG-RAST pipeline. The green ones are embarrassingly parallel, which can be split into multiple workunits by AWE.

We have deployed the Shock and AWE systems in the Magellan system [29] at Argonne. Magellan is a large-scale cloud computing system managed by Openstack software, tuned for scientific computing workloads. Magellan consists of 7,500 compute cores, 30 TB of RAM, and about 1 PB of storage, connected with a QDR InfiniBand fabric. Magellan at Argonne has been serving multiple research projects including MG-RAST.

We conducted experiments using virtual machines (VMs) in the Magellan cloud. We use the default configuration for all the VMs: 8 VCPUs (each with 2.6 GHz), 22 GB memory, and 300 GB disk. Shock has an additional 10 TB attached volumn to store data. The Shock and AWE servers have special ports open for RESTful APIs. In the experiments, we have another 80 VM quota for AWE clients, meaning that we can dynamically start and terminate VMs for AWE clients and can run up to 80 AWE clients simultaneously. The file transfer speed between AWE clients and Shock can be up to 110 MB per second.

B. Sample Jobs

We selected samples from MG-RAST job repository representing different file formats and sample sizes (Table 1).

TABLE I
Job characteristics

Format	Name	Size (Mbp)	Seqs (M)	Avg. len.	File Size
FASTA [3]	FA1	100	1.0	100	133 MB
	FA2	500	1.0	498	587 MB
	FA3	1000	10	100	1.36 GB
	FA4	2000	20	98	2.16 GB
	FA5	5000	51	98	8.15 GB
FASTQ [11]	FQ1	100	0.99	100	337 MB
	FQ2	500	50	100	1.59 GB
	FQ3	1000	7.8	128	2.52 GB
	FQ4	2000	27	75	4.63 GB
	FQ5	5000	28	177	11.0 GB

For sample, we run the MG-RAST pipeline for protein analysis (marked as 1, 2, 4, 5, 6, 7 in Figure 4), which comprises the most expensive tasks on the critical path. The tasks marked in green can be parallelized or split (i.e., screen, gene calling, and similarity search). Each run of the experiment is called a job, or a test case. In the experiments we run the jobs with different splitting levels (number of workunits to split) and with different numbers of client quota (the maximum AWE clients we can run simultaneously).

C. Evaluation Results

1) Individual jobs: To evaluate the speedup brought by parallelizing task execution, we conduct experiments that run each sample with different splitting levels: the number of workunits to split for parallelizable tasks. Specifically, we set the split level to 1, 2, 4, 8, 16, and 32 (denoted as w1 to w32).

Figure 5 shows the runtime of all jobs at different splitting levels. The job runtime comprises the actual program execution time and data movement time. As shown in the figure, every job can benefit from splitting tasks. The serial execution (w1) can be reduced from more than 50 hours to under 10 hours, or from under 10 hours to under 1 hour (meaning up to 90% time reduction). We observe that smaller jobs (regarding the input data size) can run longer than bigger ones do (e.g., FA2 vs FA3, FQ2 vs FQ3, FQ4 vs FQ5). The reason is that the total runtime is not determined by the raw input, which is the input data only for the first-stage task (preprocessing). In

fact, it is largely dependent on the input data size for the most expensive tasks (i.e., protein identification based on Blat [17], a tool for sequence similarity search).

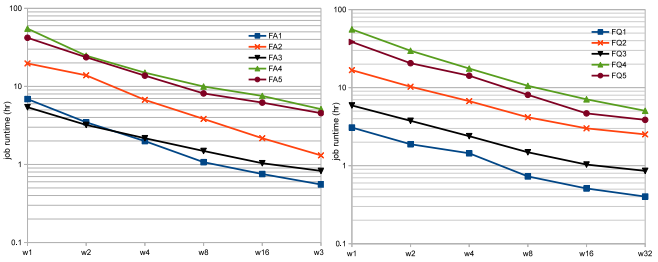

Fig. 5. Job running time (in hours)

2) Multijob workloads: To measure system throughput variation, we conduct following experiments: run the 10 jobs (in Table 1) as a whole (submit them all at the beginning) under different resource configurations, and measure the timespan (time between first job submission and the last job completion). Since the total work delivered is the same, a shorter timespan corresponds to a higher throughput.

Figure 6 shows the total timespan of workloads running under different configurations, represented by number of client quota (C) and number of split workunits (W). The client quota restrains the maximum number of workunits running in parallel.

As shown in the figure, both increasing C and W can reduce the timespan. Specifically, when the client quota is 40, increasing split workunits from 8 to 16 results in significant running time reduction (20%), while at the same quota, further increasing W to 32 does not further increase throughput obviously. This indicates that with 40 clients, splitting tasks into 8 workunits does not fully utilize the client quota, while splitting into 16, the system is almost saturated with workunits. At this condition, increasing client quota can further increase the throughput. As shown in the figure, with 80 clients, w16 and w32 can be reduce timespan by 20% and 34% compared the cases with client quota 40.

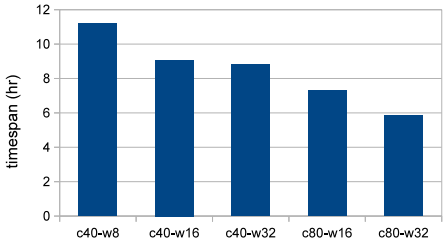

Fig. 6. Total timespan of different settings. C (40/80): number of client quota; W (8/16/32): number of split workunits per task

Figure 7 shows the data movement overhead of jobs in each of the throughput tests. The overhead of a job is calculated as the total data movement time of all workunits divided by the total runtime of all workunits. The average, maximum, and minimum values in the figure are among the 10 jobs in each test run. The range of overhead is moderate, but it increases as number of client quota increases. It is because AWE clients are competing the bandwidth to the single Shock server. We can address this problem by deploying multiple Shock, each serving a group of AWE clients.

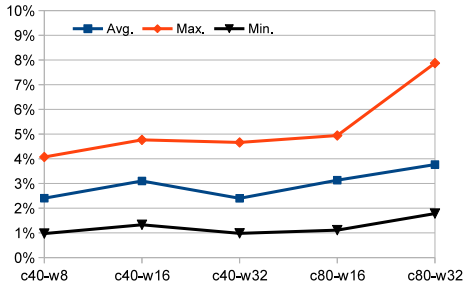

Fig. 7. Data movement overhead

IV. Related Work

A number of available scientific workflow systems have been developed for different needs [9], such as GUI-based Taverna [21] and Kepler [18], and script-based Pegasus [13] and Swift [30]. In order to utilize high-performance computing resources, existing workflow systems typically submit jobs to external resource management system such as HTCondor [28] and SGE [8]. Our platform not only provides workflow management but also serves as a resource management system providing capability of queue management and computing clients management. Also, our platform integrates workflow management and data management in a loosely coupled fashion (using REST APIs) that facilities the computation and management of large data sets. Therefore, our system is an integrated data analysis ecosystem including task, computing resource, and data management.

Over the past decade, bioinformatics has been utilizing distributed systems, such as grids [25] and clouds [22][24]. Our framework utilizes multiple techniques: the tasks can be executed on distributed, heterogeneous computing resources including supercomputers in data centers or virtual machines in clouds. Our previous work [31][33] implemented an approach to scale MG-RAST tasks into the Magellan cloud, but it is for one computationally expensive task only (not supporting the full pipeline). Our new infrastructure is much more comprehensive, which can be used to build a general data analysis ecosystem providing workflow, resource, and data management.

Emerging programming models such as MapReduce [12] also applied to bioinformatics [19]. The way AWE splits and merges data looks similar to MapReduce, but the compute diagrams are different in several aspects. From the system perspective, MapReduce requires that all the data be stored in the HDFS file system in a cluster. AWE workers do not

need to be in the same file system; instead, they can be run on geographically distributed, heterogeneous computing resources. From the workload perspective, AWE is suitable for high-compute-to-I/O workloads (a small amount of data needs a long time to compute) which is typically true for (meta)genomics workloads. MapReduce is more suitable for data intensive workloads where each task requires a short time to run. Also, AWE is for workflows/pipelines while MapReduce is for individual applications; hence, a MapReduce job can be dispatched as an AWE task at some pipeline stage.

V. Conclusion

With many areas of computational sciences thriving, the accompanied data deluge imposes big challenges in data management and analysis. In this paper, we address this problem from the perspective of computational biology. Specifically, we are building a scalable data analysis infrastructure to support the execution and data management of higher-level biological sequence analysis applications. As a product, we have developed the Shock data management and AWE workflow management system which can be used to build a scalable a data analysis platform. Our systems are evaluated by running the MG-RAST pipeline and is also suitable for building any kinds of pipeline for biological sequence data analysis. In the future, we plan to deploy our platform to serve more applications. From the system perspective, we plan to enhance our platform with more comprehensive resource management and jobs scheduling strategies (e.g., multisite job coscheduling [27]) to achieve optimal system throughput.

Acknowledgments

This work is supported by the DOE Systems Biology Knowledgebase (KBase) project. The work at Argonne is supported in part by the U.S. Department of Energy (DOE), Office of Biological and Environmental Research, and in part by by the DOE, Office of Science, under Contract DE-AC02-06CH11357.

References

[1] AWE project. https://github.com/MG-RAST/AWE
[2] Earth Microbiome Project. http://www.earthmicrobiome.org
[3] FASTA format. http://en.wikipedia.org/wiki/FASTA_format
[4] MG-RAST website. http://metagenomics.anl.gov
[5] MongoDB. http://www.mongodb.org
[6] NIH Human Microbiome Project. http://www.hmpdacc.org
[7] Shock project. https://github.com/MG-RAST/Shock
[8] Oracle Grid Engine. http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html
[9] A. Barker and J. Van Hemert, "Scientific workflow: A survey and research directions," *Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science* 4967:746-753, 2008.
[10] N. Blow, "Metagenomics: Exploring unseen communities," *Nature*, 453:687-690, 2008.
[11] P. Cock, C. Fields, N. Goto, M. Heuer, and P. Rice,"The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucl. Acids Res.*, 38(6):1767-1771, 2010.
[12] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. of Sixth Symposium on Operating System Design and Implementation*, 2008.
[13] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, 13:219-237, 2005.
[14] N. Desai, D. Antonopoulos, J. Gilbert, E. Glass, and F. Meyer, "From genomics to metagenomics," *Current Opinion in Biotechnology*, 23:72-76, 2012.
[15] R. Fielding and R. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, 2(2):115-150, 2002.
[16] J. Goecks, A. Nekrutenko, J. Taylor, et al., "Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life science," *Genome Biology*, 11:R86, 2010.
[17] W. J. Kent, "BLAT–the BLAST-like alignment tool," *Genome Research*, 12(4):656-664, 2002.
[18] B. Ludascher, C. Berkley, M. Jones, and E. A. Lee, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, 18(10):1039-1065, 2006.
[19] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. DePristo, "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, 20(9):1297-1303, 2010.
[20] F. Meyer, D. Paarmann, M. D'Souza, R. Olson, E. M. Glass, M. Kubal, T. Paczian, A. Rodriguez, R. Stevens, A. Wilke, J. Wilkening, and R. Edwards, " The metagenomics RAST server–a public resource for the automatic phylogenetic and functional analysis of metagenomes," *BMC Bioinformatics*, 9(386):1-8, 2008.
[21] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble, "Taverna, reloaded," In *Proc. of International Conference on Scientific and Statistical Database Management (SSDBM)*, 2010.
[22] N. Mohamed, H. Lin, and W.-C. Feng, "Accelerating data-intensive genome analysis in the cloud," In *Proc. of International Conference on Bioinformatics and Computational Biology (BICoB)*, 2013.
[23] D. Pushkarev, N. Neff, and S. Quake, "Single-molecule sequencing of an individual human genome," *Nature Biotechnoly*, 27:847-850, 2009.
[24] J. Qiu, J. Ekanayake, T. Gunarathne, J. Y. Choi, S.-H. Bae, H. Li, B. Zhang, T.-L. Wu, Y. Ruan, S. Ekanayake, A. Hughes, and G. Fox, "Hybrid cloud and cluster computing paradigms for life science applications," *BMC Bioinformatics*, 11(12):53, 2010.
[25] L. Ramakrishnan, M. S. Reed, J. L. Tilson, and D. A. Reed, "Grid portals for bioinformatics," in *International Workshop on Grid Computing Environments*, 2006.
[26] S. Sun, J. Chen, W. Li, I. Altinatas, A. Lin, S. Peltier, K. Stocks, E. Allen, M. Ellisman, J. Grethe, and J. Wooley, "Community cyberinfrastructure for advanced microbial ecology research and analysis: The CAMERA resource," *Nucleic Acids Research*, 2010. doi: 10.1093/nar/gkq1102
[27] W. Tang, N. Desai, V. Vishwanath, and Z. Lan, "Multi-domain job coscheduling on leadership computing systems," *The Journal of Supercomputing*, 63(2):367-384, 2013.
[28] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience,"*Concurrency and Computation: Practice and Experience*, 17(2-4): 323–356, 2005.
[29] U.S. Department of Energy. The Magellan report on cloud computing. Available at: http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf
[30] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, 37(9):633-652, 2011.
[31] A. Wilke, J. Wilkening, E. Glass, N. Desai, and F. Meyer, "An experience report: porting the MG-RAST rapid metagenomics analysis pipeline to the cloud," *Concurrency and Computation: Practice and Experience*, 23:2250-2257, 2011.
[32] A. Wilke et al., "MG-RAST technical report and manual for version 3.3.6–rev.1," ftp://ftp.metagenomics.anl.gov/data/manual/mg-rast-tech-report-v3_r1.pdf
[33] J. Wilkening, A. Wilke, N. Desai, and F. Meyer, "Using clouds for metagenomics: A case study," in *Proc. of IEEE International Conference on Cluster Computing*, 2009.